Bandwidth-centric allocation of independent tasks on heterogeneous platforms

Olivier Beaumont, Larry Carter, Jeanne Ferrante, Arnaud Legrand, Yves Robert

Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON n° 5668

Bandwidth-centric allocation of independent tasks on heterogeneous platforms

Olivier Beaumont,
Larry Carter,
Jeanne Ferrante,
Arnaud Legrand and
Yves Robert

June 2001

Research Report N° 2001-25

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : lip@ens-lyon.fr

Bandwidth-centric allocation of independent tasks on heterogeneous platforms

Olivier Beaumont,
Larry Carter,
Jeanne Ferrante,
Arnaud Legrand and
Yves Robert

June 2001

Abstract

In this paper, we consider the problem of allocating a large number of independent, equal-sized tasks to a heterogenerous "grid" computing platform. Such problems arise in collaborative computing efforts like SETI@home. We use a tree to model a grid, where resources can have different speeds of computation and communication, as well as different overlap capabilities. We define a base model, and show how to determine the maximum steady-state throughput of a node in the base model, assuming we already know the throughput of the subtrees rooted at the node's children. Thus, a bottom-up traversal of the tree determines the rate at which tasks can be processed in the full tree. The best allocation is *bandwidth-centric*: if enough bandwidth is available, then all nodes are kept busy; if bandwidth is limited, then tasks should be allocated only to the children which have sufficiently small communication times, regardless of their computation power.

We then show how nodes with other capabilities — ones that allow more or less overlapping of computation and communication than the base model — can be transformed to equivalent nodes in the base model. We also show how to handle a more general communication model.

Finally, we present simulation results of several demand-driven task allocation policies that show that our bandwidth-centric method obtains better results than allocating tasks to all processors on a first-come, first serve basis.

Keywords: heterogeneous computer, allocation, scheduling, grid, metacomputing.

Résumé

Dans ce rapport, nous nous intéressons au problème de l'allocation d'un grand nombre de taches indépendantes et de taille identiques sur des plateformes de calcul hétérogènes comme la fameuse *computing grid*. Ce type de problèmes apparaît dans des projets de *metacomputing* tels que SETI@home. Nous utilisons des arbres pour modéliser les plateformes dont les ressources peuvent avoir des vitesses de calcul ou de communication différentes les unes des autres ainsi que diverses possibilités de recouvrement. Nous définissons un modèle un modèle de base et montrons comment déterminer le débit maximal d'un nœud en régime stationnaire dans ce modèle à partir du débit maximal de ses fils. Ainsi, en partant des feuilles et en remontant vers la racine, il est possible de calculer la vitesse à laquelle les taches peuvent être traitées sur cet arbre. La meilleure allocation est *bandwidth-centric*: si les bandes passante entre le père et ses fils sont suffisamment élevées alors tous les nœuds peuvent être utilisés à plein régime; dans le cas contraire, les taches doivent êtres allouées en priorité aux fils dont le temps de communication avec le père est le plus petit, sans tenir compte de leur puissance de calcul.
Nous montrons ensuite comment les nœuds ayant d'autres capacités de communications –ceux qui permettent plus ou moins de recouvrement de calcul par les communications que dans le modèle de base– peuvent être transformés en nœuds équivalents dans le modèle de base. Nous montrons également comment traiter le cas d'un modèle de communications plus général.
Enfin, nous présentons des simulations de différentes politiques d'allocations de tâches *à la demande* qui montrent que notre approche *bandwidth-centric* est meilleure que les approches classiques (premier arrivé, premier servi).

Mots-clés: Plateformes de calcul hétérogène, allocation, ordonnancement, grilles de calcul, *metacomputing*

Bandwidth-centric allocation of independent tasks on heterogeneous platforms*

Olivier Beaumont[+], Larry Carter[++], Jeanne Ferrante[++],
Arnaud Legrand[+] and Yves Robert[+]

(+) LIP, UMR CNRS–ENS Lyon–INRIA 5668
Ecole Normale Supérieure de Lyon, 69364 Lyon Cedex 07, France
e-mail: Firstname.Lastname@ens-lyon.fr

(++) Computer Science Department, University of California, San Diego
9500 Gilman Drive, La Jolla, California 92093-0114 U.S.A.
{carter,ferrante}@cs.ucsd.edu

19th June 2001

*The work of Larry Carter and Jeanne Ferrante was performed while visiting LIP.

1 Introduction

In this paper, we deal with the problem of allocating a large number of independent, equal-sized tasks to a heterogeneous "grid" computing platform. We allow a quite general model of computation, where the various resources that make up the grid can have not only different computation and communication speeds, but can even have different capabilities in terms of how much various communication tasks can be overlapped with computation. Our model is motivated by problems that are addressed by collaborative computing efforts such as SETI@home [32], factoring large numbers [13], the Mersenne prime search [31], and those distributed computing problems organized by companies such as Entropia [16]. However, such efforts must also deal with dynamically changing computational resources, which are not addressed directly in our model. We discuss in Section 8 the implication of our work for such enviroments.

We model a collection of heterogeneous resources and the communication links between them as the nodes and edges of an undirected graph. We assume that between any pair of nodes, there is only one path, that is, the graph is acyclic. Each node is a computing resource (a processor, or a cluster, or whatever) capable of computing and/or communicating with its neighbors at (possibly different) rates.

Given n independent, identical tasks to be allocated on the grid, we assume that their data is initially located on (or generated by) a single node. We make this node the root of the graph, which, being acyclic, is now a tree. The root processor decides which tasks to execute itself, and how many tasks to forward to each of its children. Each child faces in turn the same dilemma: determine how many tasks to execute, and how many to delegate to each child. Due to heterogeneity, the children may receive different amounts of work.

In this paper, we show how each node *locally* can attain the best allocation of tasks to resources that maximizes the steady-state throughput, or tasks processed per unit time, throughout the tree. We also show that this best allocation is *bandwidth-centric*: if enough bandwidth is available to the node, then all children are kept busy; if bandwidth is limited, then tasks should be allocated only to children which have sufficiently fast communication times, in order of fastest communication time. Counter-intuitively, the maximum throughput in the tree is achieved by delegating tasks to children as quickly as possible, and not by seeking their fastest solution.

We investigate several operation models: with/without computation and communication overlap, and single-port vs. multiple-port communication links. For each model, we provide the best allocation of tasks to processors, i.e. the allocation that maximize the steady-state throughput. By ignoring the processor idle time at the

start and end of task execution, we are able to give simple allocation strategies. Since the execution time will be within an additive constant of the optimal, this approach is appropriate when n is suitably large. Note that most variants of the problem of minimizing execution time are NP-complete [10, chapter 4], hence the need to restrict to the simpler optimization problem of maximizing the steady-state throughput.

We also present simulation results of demand-driven task allocation. We simulated three strategies: one that allocates tasks to all children, a second that allocates tasks only to those children specified by our bandwidth-centric approach, and a third that adjusts the priorities to match the optimal strategy. These results show that in the case of limited bandwidth, the latter two approach not only use fewer processors, but also result in execution times that are often significantly faster.

In Section 2 we discuss some issues concerning the modeling of heterogeneous platforms, and define several models of communication and computation. We give in Section 3 a simple algorithm that finds the optimal allocation of tasks in one of these models, the *base* model. Section 4 shows how to reduce the allocation problem in any of the other models to a problem in the base model. In Section 5, we show how to incorporate a more general cost model for communication; Section 6 gives some simulation results; Section 7 describes related work, and finally we give some remarks and conclusions in Section 8.

2 Modeling a grid

We first present our motivation for using a tree to model a grid, and then present the models, including our base model, used in this paper.

2.1 Motivation and background

It is natural to model a grid as an undirected graph, where edges represent two-way communiction links between grid nodes. For example, the resources available to an application may be connected through a Wide-Area Network (WAN), where gateways to the WAN are connected to a Local-Area Network (LAN) of workstations and PC's.

Often the actual interconnection structure in a given administrative domain is acyclic, and therefore is an (unrooted) tree. Examples include a cluster of workstations, or the multiple clusters that make up a computer science department's computational resources. As well, the structure is often hierarchical, as in a campus-

wide or business-wide network connecting several LAN's. In this paper, we use a tree to model the grid infrastructure. This is despite the fact that in large-scale networks such as the Internet, there are multiple paths between nodes. We return to this latter point shortly.

We also make the assumption that the data for the computation initially resides at a single node of the tree, and the results of the computation will be returned to that same node. This directly models the situation where data resides in a single data repository; it also models the situation where a scientist creates a computational model on a workstation, then farms out the computation to a variety of resources available. We designate the node that serves as the source of the data the root of the tree. Thus, work to be done is initiated by communication from parents towards their children, and results are passed from children to its parent.

The advantage of using trees, as opposed to the more general graphs, in our modeling, include

- No choices need be made about about how to route data.
- Trees fit in well with a number of programming paradigms such as Master-Worker, Remote Procedure Call, and Divide-and-Conquer. With MPI and PVM operations, even though processes commuicate point-to-point, there is still a single processor that initiates the set of processes and allocates them to processors. It is natural to think of this initiator as the parent of the child processors.

There are still important choices to be made in using a tree to model a grid. When a collection of resources are connected by a high bandwidth network, e.g. Myrinet or Ethernet, we can either make a supernode representing the collection, whose children are the actual resources, or we can choose one of the resources to be the parent, and the remainder its children. Depending on the circumstances, either might be the right choice. For example, if there is a LAN where one specific node serves as a gateway to a larger network which provides the data to the nodes in the LAN, then the latter method seems desirable. Alternatively, the best way to model the Internet's backbone is as a single supernode, and each site connected to the backbone is modeled as a child of the supernode. Our modeling technique allows us to characterize each connection by the bandwidth it experiences to the backbone. This allows us to use a tree structure for the incredibly complex Internet.

It is important in modeling to use the bandwidth that the application actually achieves, rather than the theoretically achievable bandwidth. For instance, even

though there may be an OC3 connection between two supercomputer centers, the bandwidth that any given application experiences is likely to be much lower than 155 Mbit/sec. To accurately determine the bandwidth experienced by an application, a tool such as the Network Weather Service [35] or Remos [29] can be used.

Given that we have modeled each computational resource as a node, in a heterogeneous environment, there are many different factors that can be used to describe the power of that node. Later in this section, we introduce a number of parameters, representing both computational and communication speeds, as well as assumptions about what operations can be performed concurrently. In Section 5, we also introduce refinements in the communication cost model along the lines of overhead and gap from the LogP model [15].

Previous work has also suggested using a tree to model underlying computational and communication resources. In discussing the fat tree model, Leiserson gave a procedure for modeling an arbitrary computational resource (which could be a single VLSI chip or an entire supercomputer) by a tree [28]. Starting with a graph whose nodes represent processors and whose edges represent communication links, the procedure is to partition the graph into two pieces with roughly equal computational power. When this method is applied recursively, the result is a binary tree where all the processors are at the leaves.

Trees are also used by Alpern et al. as the basis of their PMH model [1]. Here, starting with the same underlying graph, nodes that are connected by a link whose bandwidth is above a certain threshhold are coalesced into supernodes. When this is applied recursively, the result is a tree whose structure depends on the threshholds.

Both of these models were developed to model homogeneous computational resources. Each can be applied as well in the heterogeneous case, but it remains an open question as to the accuracy of the resulting model.

Yet another model of a heterogeneous network of computing resources is the *effective network view* (ENV) model. The procedure described in [34] for constructing such a model from measurements of a system actually constructs a graph that may not be a tree. However, the method could easily be modified to only produce trees.[1]

2.2 Our models

The target architectural/application framework is represented by a node-weighted edge-weighted tree $T = (V, E, w, c)$ as illustrated in Figure 1. Each node $P_i \epsilon V$

[1] This would be accomplished by combining nodes that represent alternate networks, rather than leaving them as separate nodes, in the optional "third active test" phase.

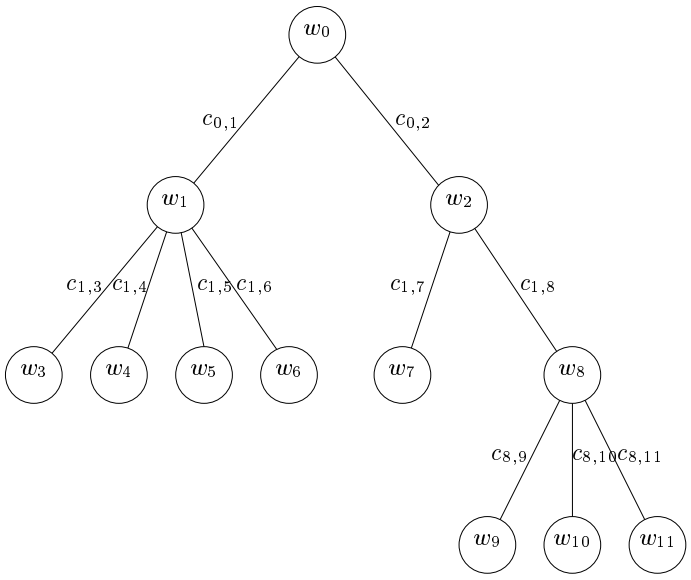

Figure 1: A tree labeled with node (computation) and edge (communication) weights.

represents a computing resource of weight w_i, meaning that node P_i requires w_i units of time to execute one task. Each edge $e : P_i \to P_j$ is labeled by a value c_{ij} which represents the time needed by node P_i (the one closer to the root) to communicate the data for one task to node P_j (one of its children) plus the time for the child to return the result when it is finished. For the purpose of computing steady-state behavior, it does not matter what fraction of the commuication time is spent sending a problem and what fraction is spent receiving the results.[2] To simplify the exposition, we will henceforth assume that all the time is spent sending the task's data to the child, and no time is needed to communicate the results back.

The only assumptions on the values w_i and c_{ij} we make are that they are integers with $w_i > 0$ and $c_{ij} \geq 0$. We disallow $w_i = 0$, since it would permit a node to perform an infinite number of tasks, and we restrict ourselves to integers (or equivalently to rational numbers) since with arbitrary real numbers, it could happen that no periodic could be within an additive constant of the optimal schedule asymptotically.[3]

[2] If our goal was to compute the total execution time of an application, this distinction might be important.

[3] For example, if $c_{01} = 1 - \epsilon$ where ϵ is an irrational number less than .5, and $w_1 = w_2 = c_{02} = 1$, then there is a schedule that in t timesteps executes $t(1+\epsilon) - o(1)$ tasks. But a periodic schedule can execute at most $t(1+r)$ tasks, where r is a rational number less than ϵ.

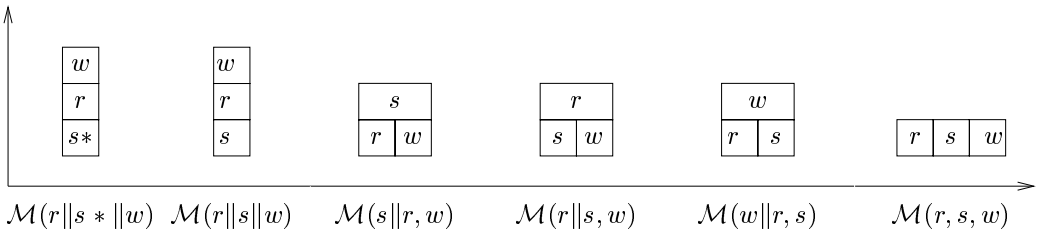

Figure 2: Classification of the operating models.

There are several scenarios for the operation of the processors, which we describe from the most powerful to the least hardware-demanding: see Figure 2, where "r" stands for *receive*, "s" stands for *send* and "w" stands for *work*, i.e. *compute*. In the figures, when two squares are placed next to each other horizontally, it means that only one of them can be accomplished at a time, while vertical placement is used to indicate that concurrent operation is possible. We also use "$\|$" (respectively ",") to indicate parallel (sequential) order of operations in the models.

$\mathcal{M}(r\|s*\|w)$**: Full overlap, multiple-port** In this first model, a processor node can simultaneously receive data from its parent, perform some (independent) computation, and send data to all of its children. This model is not realistic if the number of children is large.

$\mathcal{M}(r\|s\|w)$**: Full overlap, single-port** In this second model, a processor node can simultaneously receive data from its parent, perform some (independent) computation, and send data to one of its children. At any given time-step, there are at most two communications taking place, one from the parent and/or one to a single child. This model is representative of a large class of modern machines, and is our *base model*.

$\mathcal{M}(r\|s,w)$**: Receive-in-Parallel, single-port** In this third model, as in the next two, a processor node has one single level of parallelism: it can perform two actions simultaneously. In the $\mathcal{M}(s\|r,w)$ model, a processor can simultaneously receive data from its parent and either perform some (independent) computation, or send data to one of its children. The only parallelism inside the node is the possibility to receive from the parent while doing something else (either computing or sending to one child).

$\mathcal{M}(s\|r,w)$**: Send-in-Parallel, single-port** In this fourth model, a processor node can simultaneously send data to one of its children and either perform some (independent) computation, or receive data from its parent. The only parallelism inside the node is the possibility to send to one child while doing something else (either computing or receiving from the parent).

$\mathcal{M}(w\|r,s)$**: Work-in-Parallel, single-port** In this fifth model, a processor node can simultaneously compute and execute a single communication, either sending data to one of its children or receiving data from its parent.

$\mathcal{M}(r,s,w)$**: No internal parallelism** In this sixth and last model, a processor node can only do one thing at a time: either receiving from its parent, or computing, or sending data to one of its children. This is really the low-end computer!

There are n independent tasks to be executed on the tree. Each task requires w_i units of time when executed by node P_i. So the smaller w_i, the faster the processor node P_i.

Initially, the data for all tasks is stored in the root processor. It takes c_{ij} units of time for node P_i to send (the data for) one task to its child P_j. In other words, if a task is resident on node P_i at time-step t, and if P_i initiates the communication to P_j at that time-step, then the task will be available to P_j at time $t + c_{ij}$. Tasks are atomic, their computation or communication cannot be preempted. A task represents the granularity of the application.

Given a weighted tree and an operation mode (any one of the six listed above) for each node, one might want to determine how many tasks should be executed by each node so that the overall computation time is minimized. This is a difficult problem, and in fact cannot be answered without knowing more about the network than is in our model, such as the network latencies. Instead, we will answer a related question. As is well-known, a computation consists of a *start-up* or *initialization* interval where some processors are not running at the full speed that can be sustained, then a *steady-state* interval where all processors are running at the maximum speed that the network can sustain, and a *clean-up* interval when some but not all processors are finished. See Figure 3 for an illustration. During the steady-state interval, the operation of the tree is periodic, with b tasks executed every t time units. Both the start-up and clean-up times are bounded, no matter how many tasks there are. The question we will answer is what is the rate or *throughput*, $R = b/t$, of processing tasks in the steady-state interval. This answer can be used to give the execution time of the full application within an additive constant.

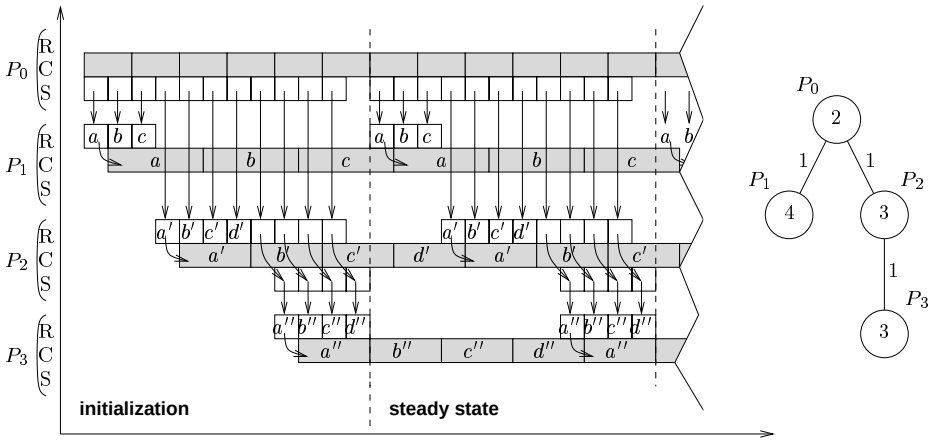

Figure 3: Initialization vs. steady state.

In steady state, the tree operates as a single (virtual) node whose computational weight $w_{\text{tree}} = \frac{t}{b} = \frac{1}{R}$ is to be minimized. The major contribution of this paper is to determine the best allocation of tasks to processors, according to the criterion of minimizing w_{tree}, for any input tree and any of the previously listed operating modes.

3 Solution for the Base model

In this section, we show how to compute the best task allocation using the base model $\mathcal{M}(r\|s\|w)$, where a processor node can simultaneously receive data from its parent, send data to one of its children and perform some (independent) computation.

3.1 Fork graph

We start with simple fork graphs before dealing with arbitrary tree graphs. A fork graph, as shown in Figure 4, consists of a node P_0 and its k children $P_1 \dots P_k$. In the base model, P_0 can communicate with a single child at a time: it needs c_i units of time to communicate a task to child P_i. Concurrently, P_0 can receives data from its own parent, say P_{-1}, requiring c_{-1} time per task. We give three examples in Figures 5, 6 and 7: in the first, all children operate at full rate, and in the latter two, the communication bandwidth is the limiting factor.

We now state and prove our main theorem:

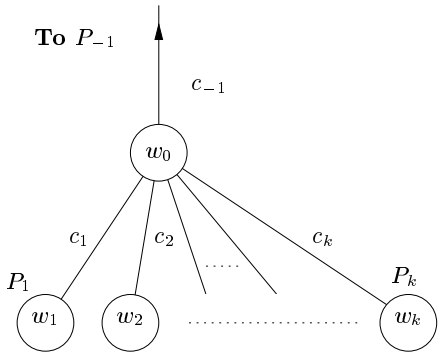

Figure 4: Fork graph.

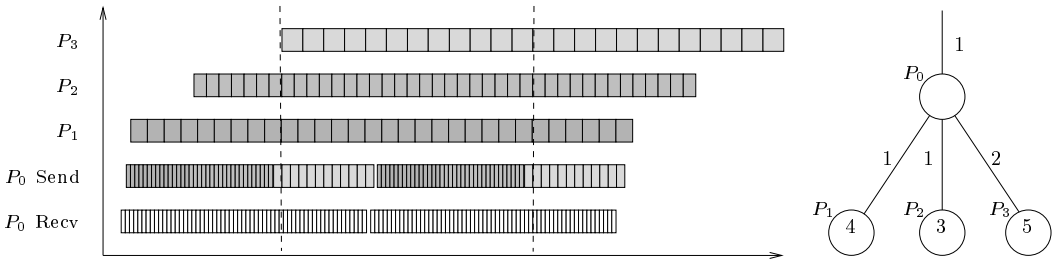

Figure 5: First example without saturation of the communication bandwidth: all children can be kept fully active

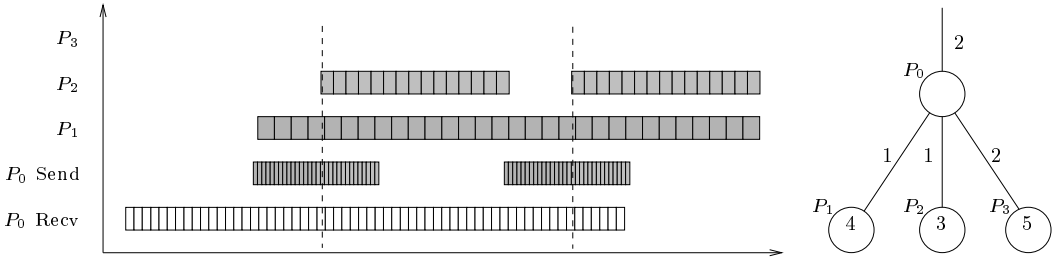

Figure 6: Second example with saturation of the communication bandwidth: some children are partially idle due to the low bandwidth between P_0 and its parent

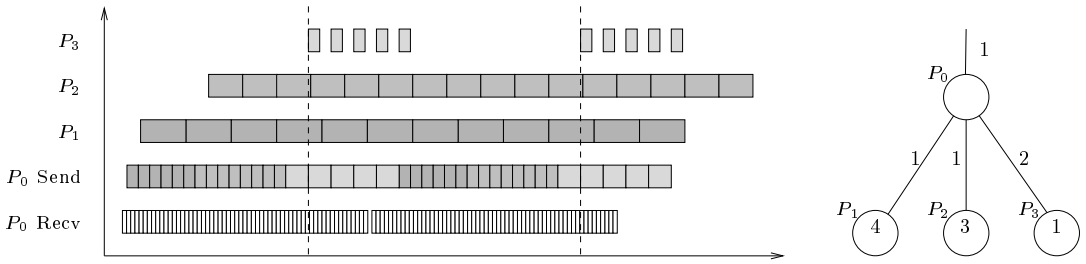

Figure 7: Third example with saturation of the communication bandwidth: child P_3 is partially idle due to its high computation speed

Proposition 1 *With the above notations, the minimal value of w_{tree} for the fork graph is obtained as follows:*

1. *Sort the children by increasing communication times. Re-number them so that $c_1 \leq c_2 \ldots \leq c_k$.*
2. *Let p be the largest index so that $\sum_{i=1}^{p} \frac{c_i}{w_i} \leq 1$. If $p < k$ let $\epsilon = 1 - \sum_{i=1}^{p} \frac{c_i}{w_i}$, otherwise let $\epsilon = 0$.*
3. *Then*

$$w_{tree} = \max\left(c_{-1}, \quad \frac{1}{\frac{1}{w_0} + \sum_{i=1}^{p} \frac{1}{w_i} + \frac{\epsilon}{c_{p+1}}}\right)$$

Intuitively, the processors cannot consume more tasks than sent by P_{-1}, hence the first term of the maximum, i.e. c_{-1}. For the second term, when $p = k$ the result is expected: it basically says that children can be fed with tasks fast enough so that they are all kept computing steadily. However if $p < k$ the result is surprising: in the situation when the communication bandwidth is limited, some children will partially starve: these are those with slow communication rates, whatever their processing speeds. In other words, a slow processor with a fast communication link is to be preferred to a fast processor with a slow communication link.

Proof Suppose the optimal periodic schedule has a fixed, steady state pattern of period T: starting at time-step t_0, the whole pattern of computations and communications repeats every T time-units, i.e. at time-step $t_0 + T$, $t_0 + 2T$, and so on. At the end of the proof, we will give upper bounds on t_0 and T for (one particular) asymptotically optimal schedule.

During this period of T units of time, let x_{-1} denote the amount of time the node spends receiving tasks from its own parent P_{-1}, x_0 be the time spent computing, x_1 be the time sending tasks to child P_1, x_2 units of time sending tasks to child $P_2, \ldots,$ and x_k units of time sending tasks to child P_k. Because of periodicity and integral costs, all of the x_i's are integers.

Because of the overlap hypothesis, the computation of x_0 tasks by P_0 doesn't diminish either the number x_{-1} it can receive from P_{-1} nor $x_1, x_2, \ldots, x_k$. We have the following constraints for x_0: $0 \leq x_0 \leq T$, x_0 should be a multiple of w_0. The number of tasks that will be processed by P_0 is $\frac{x_0}{w_0}$.

We have similar constraints on x_{-1}: $0 \leq x_{-1} \leq T$, x_{-1} should be a multiple of c_{-1}. The number of tasks that are sent to P_0 is $\frac{x_{-1}}{c_{-1}}$.

There are more constraints on $x_1, \ldots, x_k$. For $1 \leq i \leq k$, $0 \leq x_i \leq T$, and x_i should be a multiple of c_i. The single-port hypothesis translates into $x_1 + x_2 + \ldots + x_k \leq T$. The number of tasks received by child P_i is $\frac{x_i}{c_i}$; it can process these tasks within the time frame iff $\frac{x_i}{c_i} \times w_i \leq T$. Note that P_i should not receive more tasks than it can process.

There is a final constraint that ensures the steady-state of the whole process: the number of tasks sent to the fork-join graph, i.e. $\frac{x_{-1}}{c_{-1}}$, should be equal to the number of tasks that it can consume, i.e. $\frac{x_0}{w_0} + \sum_{i=1}^{k} \frac{x_i}{c_i}$. This last quantity is exactly the quantity that we want to maximize.

Let us summarize the situation as a linear programming problem, where the objective function is the number of tasks consumed within the T units of time:

$$\textbf{Maximize } \frac{x_0}{w_0} + \sum_{i=1}^{k} \frac{x_i}{c_i}$$
$$\textbf{subject to } \begin{cases} 0 \leq x_i \leq T \text{ for } -1 \leq i \leq k \\ \frac{x_i}{c_i} \times w_i \leq T \text{ for } 1 \leq i \leq k \\ \sum_{i=1}^{k} x_i \leq T \\ \frac{x_{-1}}{c_{-1}} = \frac{x_0}{w_0} + \sum_{i=1}^{k} \frac{x_i}{c_i} \end{cases}$$

Because everything is linear in T, we can normalize the problem and set $T = 1$: we now look for nonnegative *rational* values $x_{-1}, x_0, x_1, \ldots, x_k$ such that

$$\begin{array}{l} x_{-1} \leq 1 \\ x_0 \leq 1 \\ \sum_{i=1}^{k} x_i \leq 1 \\ x_i \leq \frac{c_i}{w_i} \text{ for } 1 \leq i \leq k \\ \frac{x_{-1}}{c_{-1}} = \frac{x_0}{w_0} + \sum_{i=1}^{k} \frac{x_i}{c_i} \end{array}$$

We can further normalize these equations by introducing the rates R_i: $R_{-1} = \frac{x_{-1}}{c_{-1}}$ is the rate of tasks per second received from the parent P_{-1}, $R_0 = \frac{x_0}{w_0}$ is the rate they are executed in the parent node P_0, and R_i, for $1 \leq i \leq k$, is the rate they are sent to and executed on the i-th child. We obtain the following formulation of the problem of determining the optimal task allocation for the base model:

Base Problem: Maximize $\sum_{i=0}^{k} R_i$, **subject to**
(B0) $R_{-1} = \sum_{i=0}^{k} R_i$
(B1) $R_{-1} c_{-1} \leq 1$
(B2) $R_i \leq \frac{1}{w_i}$ for $0 \leq i \leq k$
(B3) $\sum_{i=1}^{k} R_i c_i \leq 1$

Note that moving from the x_i's to the rates R_i's has a technical advantage: the new formulation nicely encompasses the case where some communication time c_i is zero (which might be appropriate, for instance, for a shared-memory multiprocessor).

Let R be the solution of the Base Problem. We claim that $R = \min\left(\frac{1}{c_{-1}}, \frac{1}{w_0} + S\right)$ (unless $c_{-1} = 0$, in that case $R = \frac{1}{w_0} + S$) , where S is the solution to the following problem:

Auxiliary problem: Maximize $\sum_{i=1}^{k} R_i$, **subject to**
(i) $R_i w_i \leq 1$ for $1 \leq i \leq k$
(ii) $\sum_{i=1}^{k} R_i c_i \leq 1$

Because the auxiliary problem is less constrained than the original one, we immediately have that $\frac{1}{w_0} + S \geq R$. We also have $\frac{1}{c_{-1}} \geq R$, because of (B0) and (B1), hence $\min\left(\frac{1}{c_{-1}}, \frac{1}{w_0} + S\right) \geq R$. To show the reverse inequality, there are two cases, according to the value of $\min\left(\frac{1}{c_{-1}}, \frac{1}{w_0} + S\right)$. Assume first that $\min\left(\frac{1}{c_{-1}}, \frac{1}{w_0} + S\right) = \frac{1}{c_{-1}}$. Let $(R_1, \ldots, R_k)$ be the optimal solution of the auxiliary problem: $S = \sum_{i=1}^{k} R_i$ and $\frac{1}{w_0} + S \geq \frac{1}{c_{-1}}$. Let $\alpha = \frac{\frac{1}{c_{-1}}}{\frac{1}{w_0} + S} \leq 1$. Then $\left(\frac{1}{c_{-1}}, \frac{\alpha}{w_0}, \alpha R_1, \ldots, \alpha R_k\right)$ is a solution to the base problem whose objective function is equal to $\frac{1}{c_{-1}}$. Therefore $\frac{1}{c_{-1}} \leq R$.

Assume now that $\min\left(\frac{1}{c_{-1}}, \frac{1}{w_0} + S\right) = \frac{1}{w_0} + S$. Let $(R_1, \ldots, R_k)$ be the optimal solution of the auxiliary problem: $S = \sum_{i=1}^{k} R_i$ and $\frac{1}{w_0} + S \leq \frac{1}{c_{-1}}$. Let $R_0 = \frac{1}{w_0}$ and $R_{-1} = \sum_{i=0}^{k} R_i$. Then $(R_{-1}, R_0, R_1, \ldots, R_k)$ is a solution to the base problem (note that (B1) is satisfied because of the hypothesis). Hence $\frac{1}{w_0} + S \leq R$. This concludes the reduction to the auxiliary problem.

We can now come to the solution of the auxiliary problem. As in the statement of the theorem, let p be the largest index so that $\sum_{i=1}^{p} \frac{c_i}{w_i} \leq 1$. Let $R_i^* = \frac{1}{w_i}$ for $1 \leq i \leq p$. If $p < k$, let $R_{p+1}^* = \frac{\epsilon}{c_{p+1}}$, where $\epsilon = 1 - \sum_{i=1}^{p} \frac{c_i}{w_i}$. If $p+1 < k$, let $R_i^* = 0$ for $p+2 \leq i \leq k$. We claim that (R_i^*) is the optimal solution of the optimization problem:

- First, it is indeed a solution. We have $R_{p+1}^* \leq \frac{1}{w_{p+1}}$ when $p < k$. This comes directly from the definition of p: since $\sum_{i=1}^{p+1} \frac{c_i}{w_i} > 1$, we have $\epsilon = 1 - \sum_{i=1}^{p} \frac{c_i}{w_i} < \frac{c_{p+1}}{w_{p+1}}$, hence $\frac{\epsilon}{c_{p+1}} \leq \frac{1}{w_{p+1}}$. As for (ii), $\sum_{i=1}^{j} R_i c_i = \sum_{i=1}^{j} \frac{c_i}{w_i} \leq 1$ for all $j \leq p$, by definition of p. And if $p < k$, $\sum_{i=1}^{p+1} R_i c_i = 1$, by definition of R_{p+1}.
- Second, it is the solution that maximizes the objective function. To see this, consider all the optimal solutions, i.e. all the solutions that achieve the optimal value of the objective function. Among these optimal solutions, consider one solution (R_i^*) such that R_1 is maximal. Assume by contradiction that $R_1 < R_1^* = \frac{1}{w_1}$. Then there exists at least one index $j > 2$ such that $R_j > R_j^*$, otherwise the solution would not be optimal. Now, since the c_i are sorted, we have $c_1 \leq c_j$. We do not change the value of the objective function if we let $R_1 = R_1 + \tau$ and $R_j = R_j - \tau$, where τ is an arbitrary small nonnegative rational number. However, we do have a new solution to the optimization problem, because $(R_1 + \tau)c_1 + (R_j - \tau)c_j \leq R_1 c_1 + R_j c_j$. Hence we have an optimal solution with a larger R_1 than the original one, a contradiction. Hence we have shown that there exist optimal solutions such that $R_1 = R_1^*$. We restrict to such solutions without loss of generality and we iterate the process: we finally derive that (R_i^*) is an optimal solution.

The optimal solution of the auxiliary problem is

$$S = \sum_{i=1}^{k} R_i = \sum_{i=1}^{p} \frac{1}{w_i} + \frac{\epsilon}{c_{p+1}}$$

The optimal solution of the base problem is

$$R = \min\left(\frac{1}{c_{-1}}, \frac{1}{w_0} + S\right)$$

Finally, $w_{\text{tree}} = \frac{1}{R}$, which establishes our claim.

To conclude the proof, we only need to provide values for t_0 and T. Since we know the optimal solution (R_i^*), we choose T such that an integral number of tasks

can be processed by each processor. We have $R_i^* = \frac{1}{w_i}$ for $0 \leq i \leq p$, hence T must be a multiple of each w_i, $0 \leq i \leq p$. Since $R_{p+1}^* = \frac{\epsilon}{c_{p+1}}$, T should be a multiple of c_{p+1}. Note that $T\epsilon$ is an integer if T is a multiple of each w_i, $0 \leq i \leq p$. Finally, T should be larger than c_{-1}. A valid value (though not necessarily the smallest possible value) for T is

$$T = \text{lcm}(w_0, w_1, \ldots, w_p, c_{-1}, c_{p+1}).$$

It is not surprising that $c_1, \ldots, c_p$ do not appear in the expression for T, because the choice of ϵ ensures that $c_i \leq w_i$ for $1 \leq i \leq p$. Finally, we can construct a schedule that enters the steady-state behavior with $t_0 = T$: to see this, perform exactly the same pattern of communications but no computation at all from time-step 0 to $t_0 - 1 = T - 1$; each processor is then provided with exactly as many tasks as required to enter its steady-state.

A less formal (but much shorter) proof of Proposition 1 is the following: sorting the c_i and feeding as many tasks as possible to the children taken in that order maximizes the number of tasks that are communicated to the children, hence the number of tasks that are processed by the children. Add those processed by the parent, and take the minimum with the input rate to derive the optimal value. ■

Note that the proof in Proposition 1 is fully constructive: the number of tasks to be computed by the parent and to be sent to each child is directly computed from the optimal solution (R_i^*).

3.2 Arbitrary tree graphs

The best allocation of tasks to processor nodes is easily determined using a bottom-up traversal of the tree:

Proposition 2 *Let $G = (V, E, w, c)$ be an arbitrary tree graph. The minimal value of w_{tree} for the whole graph is obtained as follows:*

1. *Consider any sub-tree consisting of several leaves and their parent. Replace this tree with a single node whose weight is given by Proposition 1*
2. *Iterate the process until there remains a single node.*

Then the minimal value is equal to the weight of the single node.

Proof The proof is immediate: in steady state, a fork graph consisting of a parent and several leaves behaves exactly as a single node of weight determined by Proposition 1. For the root node, we can assume a link from a virtual parent with infinite capacity, i.e. $c_{-1} = 0$.

We can give (possibly unnecessarily large) bounds on the period and start-up times. The period T of an arbitrary tree is at most the least common multiple of the periods of each fork graph encountered in the bottom-up traversal of the tree. And a bound on the start-up time is this period T times the maximum depth of the tree. This time corresponds to a start-up strategy in which during the first T time units, we send a period's worth of work to the depth one nodes (but don't execute any tasks), in the next T time units we propogate this work to the depth two nodes and replenish the depth one nodes, and so on. ■

Again, the proof is fully constructive, and the optimal task allocation is computed using the bottom-up approach. Figure 8 is a small example.

4 Reductions for other models

Recall that a task allocation is specified by a non-negative set of numbers R_{-1},R_0,...,R_k, where R_{-1} is the rate of tasks per second received from the parent, R_0 is the rate they are executed in the node, and R_i, for $1 \leq i \leq k$, is the rate they are sent to and executed on the ith child. We repeat here the formulation of the problem of determining the optimal task allocation for the base model, which we refer to as the *Base* Problem[4]:

Base Problem: Maximize $\sum_{i=0}^{k} R_i$, **subject to**

(B0) $R_{-1} = \sum_{i=0}^{k} R_i$

(B1) $R_{-1} c_{-1}^B \leq 1$

(B2) $R_i \leq \frac{1}{w_i^B}$ for $0 \leq i \leq k$

(B3) $\sum_{i=1}^{k} R_i c_i^B \leq 1$

In subsequent subsections, we show how to reduce the problem of determining the optimal task allocation for each of our other models to the Base Problem, or a previously solved Problem. Each reduction will follow the same outline: we present the constraints for the model in question; then give a simple set of definitions of

[4]Note that we have superscripted the parameters of the Base model with "B", to distinguish its values from those of other models

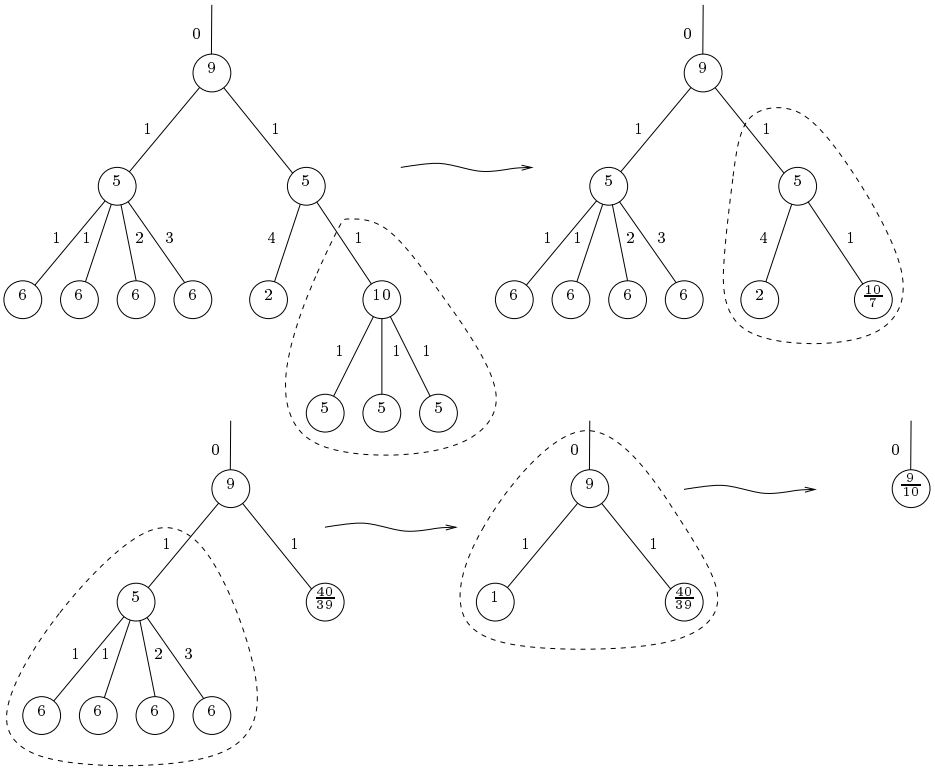

Figure 8: Computing the best allocation for a tree graph.

new variables; and finally show that by rewriting the constraints in terms of the new variables, they are exactly the constraints of an instance of the Base Problem. Thus, an optimal solution to the Base Problem provides an optimal solution to the problem in the alternate model.

4.1 Reduction of Multiport Problem to the Base Problem

In this section, we show how to transform the problem of determining the optimal task allocation for a full overlap, multiple-port model into an equivalent problem in the base model. Hence, we can use our solution for the base model to obtain a solution for the multiport model.[5]

Recall that in a full overlap, multiple-port model, $\mathcal{M}(r\|s*\|w)$, a processor node can simultaneously receive data from its father, perform some (independent) computation, and send data to all of its children; in contrast, in a base model, $\mathcal{M}(r\|s\|w)$, data can only be sent to *one* of its children.

We formulate the problem of determining the optimal task allocation for the multiport model as follows:

Multiport Problem: Maximize $\sum_{i=0}^{k} R_i$, **subject to**

(M0) $R_{-1} = \sum_{i=0}^{k} R_i$

(M1) $R_{-1} c_{-1}^M \leq 1$

(M2) $R_i \leq \frac{1}{w_i^M}$ for $0 \leq i \leq k$

(M3) $R_i c_i^M \leq 1$ for $1 \leq i \leq k$

The constraints in both problems arise from the three operations that can be executed in parallel. The first constraint (**B0** or **M0**) ensures that the number of tasks communicated per second from the parent is equal to the sum of the rate they are executed in the father node and the children; and the second constraint ensures this same quantity is bounded by the bandwidth from the parent. The third constraint is that the number of tasks per second executed by the parent (for $i = 0$) or by the i-th child (for $i > 0$) is bounded by the speed of the node. Finally, in the Multiport Problem, constraint **M3** requires that the number of tasks per second communicated to *each* child be bounded by the bandwidth to the child.

We now show that we can convert a Multiport Problem to a Base Problem by setting the communication times for the children to zero and adjusting the work times to include the original bandwith constraints.

[5] Actually, the Multiport Problem is simpler to solve than the Base Problem, but we include this transformation for completeness and to serve as a model for other reductions.

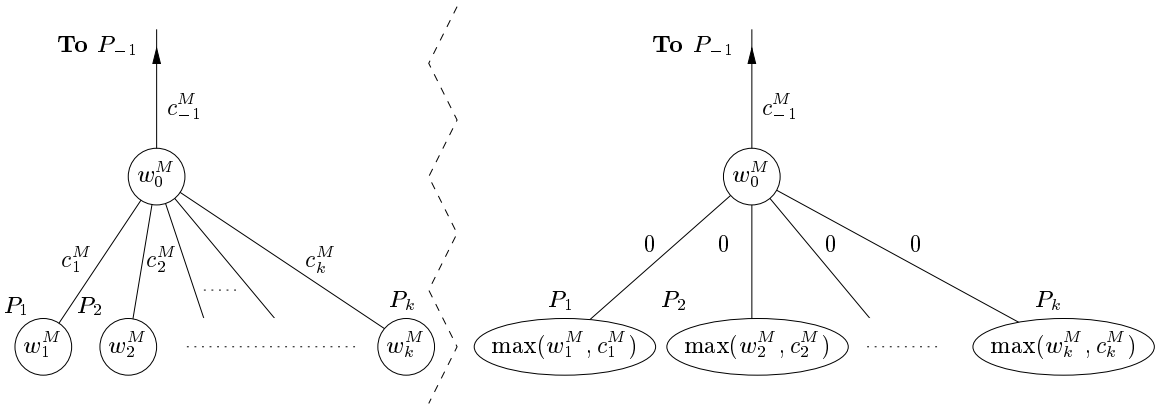

Figure 9: Reduction from Multiport to Base.

Proposition 3 *Given a Multiport Problem, there is an equivalent Base Problem whose solution yields the solution to the Multiport Problem.*

Proof Given a Multiport problem, we let $c^B_{-1} = c^M_{-1}$, $w^B_0 = w^M_0$, and for $1 \leq i \leq k$, we let $c^B_i = 0$, and $w^B_i = \max(w^M_i, c^M_i)$ (see Figure 9).

For $i = 1,...,k$, we can replace constraints **M2** and **M3** by $R_i \leq \min(\frac{1}{w^M_i}, \frac{1}{c^M_i})$, or, equivalently, $R_i \leq \frac{1}{w^B_i}$. We can also rewrite $R_0 \leq \frac{1}{w^M_0}$ as $R_0 \leq \frac{1}{w^B_0}$. Thus, the constraints **M2** and **M3** are exactly equivalent to **B2**. Obviously, **B0** and **B1** are equivalent, respectively, to **M0** and **M1**, and **B3** always holds; thus, we have shown the equivalence of the Multiport problem to the Base Problem. ■

4.2 Reduction of Work-in-Parallel Problem to the Base Problem

In this section, we show how to transform the problem of determining the optimal task allocation for $\mathcal{M}(w\|r,s)$ into an equivalent problem in the base model. Hence, we can use our solution for the base model to obtain a solution for the new problem.

Recall that in this model, $\mathcal{M}(w\|r,s)$, a processor node can simultaneously perform some (independent) computation, and either receive data from its parent or send data to one of its children; in contrast, in a base model, data can be simultaneously sent to a child or received from a parent.

We formulate the problem of determining the optimal task allocation for this model as follows:

Work-in-Parallel Problem: Maximize $\sum_{i=0}^{k} R_i^W$**, subject to**

(W0) $R_{-1}^W = \sum_{i=0}^{k} R_i^W$

(W1) $R_{-1}^W c_{-1}^W \leq 1$

(W2) $R_i^W \leq \frac{1}{w_i^W}$ for $0 \leq i \leq k$

(W3) $R_{-1}^W c_{-1}^W + \sum_{i=1}^{k} R_i^W c_i^W \leq 1$

The first three constraints **W0**, **W1** and **W2** are as in the Base and Multiport Problems. Constraint **W3** requires that the fraction of time spent communicating, either with parent or a child, be bounded by 1.

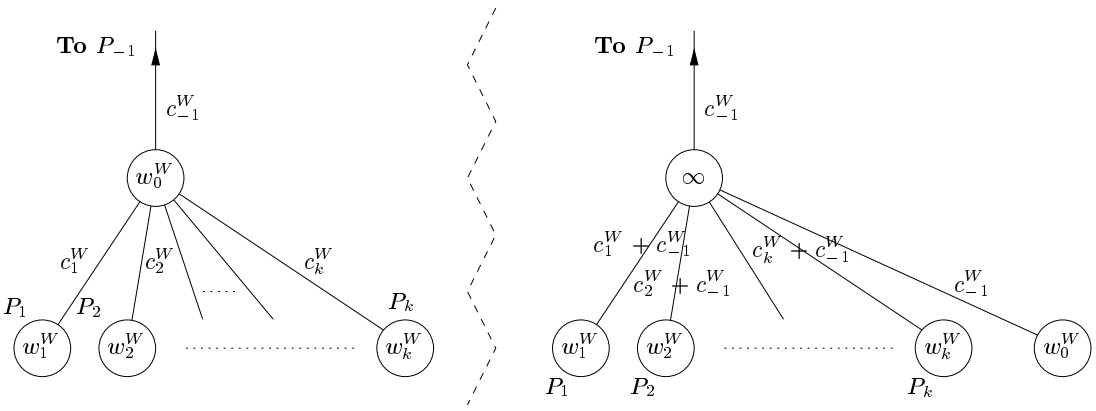

Figure 10: Reduction from Work-in-Parallel to Base.

We now show that we can convert a Work-in-Parallel Problem to a Base Problem by adding the parent's communication time to each of the children's, and by representing the node's processor as a new child.

Proposition 4 *Given a Work-in-Parallel Problem, there is an equivalent Base Problem whose solution yields the solution to the Work-in-Parallel Problem.*

Proof Given a Work-in-Parallel problem, we let $c_{-1}^B = c_{-1}^W$, $w_0^B = \infty$, $c_{k+1}^B = c_{-1}^W$, $w_{k+1}^B = w_0^W$, and for $1 \leq i \leq k$, we let $c_i^B = c_i^W + c_{-1}^W$, and $w_i^B = w_i^W$ (see Figure 10).

We can convert between solutions to the two problems by letting $R_i^B = R_i^W$, for $1 \leq i \leq k$, and $R_{k+1}^B = R_0^W$. R_0^B will be a new variable in the base model solution constrained by $R_0^B \leq \frac{1}{w_0^B}$, that is, $R_0^B \leq 0$,

Rewriting the constraints **W1** and **W2** with these values substituted, we obtain:

$$\begin{array}{ll} \textbf{(W1)} & R_{-1}^B = \frac{1}{c_{-1}^B} \\ \textbf{(W2)} & R_i^B \leq \frac{1}{w_i^B} \text{ for } 1 \leq i \leq k+1 \end{array}$$

These constraints, together with $R_0^B \leq \frac{1}{w_0^B}$, are exactly what is needed to obtain **B1** and **B2**. To obtain **B3**, we rewrite the left-hand side of **W3** as

$R_{-1}^W c_{-1}^W + \sum_{i=1}^k R_i^W c_i^W = \sum_{i=1}^k R_i^W (c_i^B - c_{-1}^B) + \sum_{i=0}^k R_i^W c_{-1}^B = \sum_{i=1}^k R_i^W c_i^B + R_0^W c_{-1}^B = \sum_{i=1}^k R_i^W c_i^B + R_{k+1}^B c_{k+1}^B = \sum_{i=0}^{k+1} R_i^B c_i^B$.

We rewrite **W0** as $R_{-1}^W = \sum_{i=0}^k R_i^W = \sum_{i=0}^{k+1} R_i^B = R_{-1}^B$, obtaining **B0**. Thus, we have shown the equivalence of the Work-in-Parallel Problem to the Base Problem. ■

Proposition 5 *If $w_0^W = \infty$, then after translating to the Base Problem using Propostion 4, the solution to the Base Problem has $R_{k+1}^B = 0$, and the maximum is obtained by the maximization of $\sum_{i=1}^k R_i^B$.*

Proof Since $w_{k+1}^B = w_0^W = \infty$, the constraint $R_{k+1}^B \leq \frac{1}{w_{k+1}^B} = \frac{1}{\infty}$ implies that $R_{k+1}^B = 0$. ■

Notice that if we had defined the Base Problem by deleting R_{k+1}^B, we would have obtained the same solution as the first Base Problem.

Corollary 1 *If a Work-in Parallel Problem with k children has $w_0^W = \infty$, then after translating to the Base Problem, the final solution has only k children.*

4.3 Reduction of No-Overlap Problem to the Base Problem

In this section, we show how to handle the cases where communication with the child cannot overlap computation. These cases are the fully sequential model $\mathcal{M}(r, s, w)$, and the model $\mathcal{M}(r \| s, w)$, where receiving from the parent can be done in parallel with either of the other operations.

We formulate the problem of determining the optimal task allocation for either of these models as follows:

No-Overlap Problem: Maximize $\sum_{i=0}^{k} R_i^N$**, subject to**

(N0) $R_{-1}^N = \sum_{i=0}^{k} R_i^N$

(N1) $R_{-1}^N c_{-1}^N \leq 1$

(N2) $R_i^N \leq \frac{1}{w_i^N}$ for $0 \leq i \leq k$

(N3) $R_0^N w_0^N + \sum_{i=1}^{k} R_i^N c_i^N + \delta(R_{-1}^N c_{-1}^N) \leq 1$
where $\delta = 1$ if the model is $\mathcal{M}(r, s, w)$, and $\delta = 0$ if the model is $\mathcal{M}(r \| s, w)$

The first three constraints **N0**, **N1** and **N2** are as in previous problems. Constraint **N3** requires that the operations that are sequentialized in each of the models be bounded by the time available.

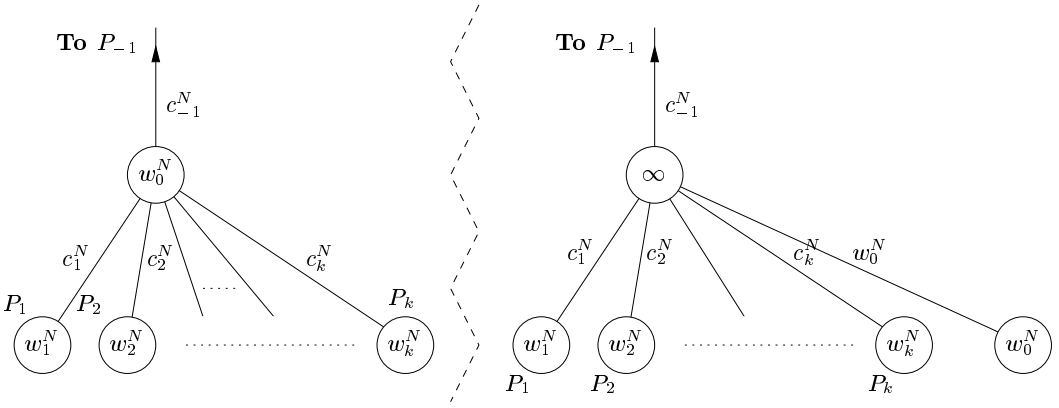

Figure 11: Reduction from No-Overlap to Base.

We now show how to convert the No-Overlap Problem to the Base Problem, again by replacing the node's processor with a new child.

Proposition 6 *Given a No-Overlap Problem, there is an equivalent problem, Work-in-Parallel in the case of* $\delta = 1$*, and Base in the case* $\delta = 0$*, whose solution yields the solution to the No-Overlap Problem.*

Proof Given a No-Overlap Problem, we let $c_{-1} = c_{-1}^N$, $w_0 = \infty$, $c_{k+1} = w_0^N$, $w_{k+1} = w_0^N$, and for $1 \leq i \leq k$, we let $c_i = c_i^N$, and $w_i = w_i^N$ (see Figure 11). As in the proof of Proposition 4, we define, for $1 \leq i \leq k$, $R_i = R_i^N$, and $R_{k+1} = R_0^N$,

and constrain the new variable R_0 by$R_0 \leq \frac{1}{w_0}$, ensuring it will be 0. Notice that $\sum_{i=0}^{k} R_i^N = \sum_{i=0}^{k+1} R_i$, (and therefore $R_{-1} = R_{-1}^N$), and $\sum_{i=1}^{k} R_i^N c_i^N + R_0^N w_0^N = \sum_{i=1}^{k+1} R_i c_i$.

The constraints we derive from **N0-N3** are:

$$\begin{array}{ll} \textbf{(0)} & R_{-1} = \sum_{i=0}^{k+1} R_i \\ \textbf{(1)} & R_{-1} c_{-1} \leq 1 \\ \textbf{(2)} & R_i \leq \frac{1}{w_i} \text{ for } 1 \leq i \leq k+1 \\ \textbf{(3)} & \sum_{i=1}^{k+1} R_i c_i + \delta(R_{-1} c_{-1}) \leq 1 \end{array}$$

To obtain **B2** (or **W2**) as in the Work-in-Parallel proof, we include the constraint $R_0 \leq \frac{1}{w_0}$.

To obtain **B3** (or **W3**) we rewrite **N3** as $R_{k+1} w_{k+1} + \sum_{i=1}^{k} R_i c_i + \delta(R_{-1} c_{-1}) \leq 1$, or $\sum_{i=1}^{k+1} R_i c_i + \delta(R_{-1} c_{-1}) \leq 1$. This latter constraint is the same as **B3** when $\delta = 0$, and as **W3** when $\delta = 1$. ■

Note that in the case $\delta = 1$, it would appear that we go from a graph with k children to $k+1$ children using the above transformation to a Work-in-Parallel Problem, and then to a graph with $k+2$ children in transforming to a Base Problem. However, the first transformation will set $w_0 = \infty$, so by Corollary 1, the second child need not be added during the second transformation.

4.4 Reduction of Send-in-Parallel Problem to the Base Problem

In this section, we show how to transform the problem of determining the optimal task allocation for $\mathcal{M}(s \| r, w)$ into an equivalent problem in the base model. Hence, we can use our solution for the base model to obtain a solution for the new problem. Unlike the previous cases, the solution to the base model problem will have to be modified slightly to provide a solution to the original (send-in-parallel) problem.

Recall that in the send-in-parallel model, a processor node can simultaneously send data to one child, and *either* receive data from its parent *or* compute. We formulate the problem of determining the optimal task allocation for this model as follows:

Send-in-Parallel Problem: Maximize $\sum_{i=0}^{k} R_i^S$**, subject to**

(S0) $R_{-1}^S = \sum_{i=0}^{k} R_i^S$
(S1) $R_{-1}^S c_{-1}^S \leq 1$
(S2) $R_i^S \leq \frac{1}{w_i^S}$ for $0 \leq i \leq k$
(S3) $\sum_{i=1}^{k} R_i^S c_i^S \leq 1$
(S4) $R_{-1}^S c_{-1}^S + R_0^S w_0^S \leq 1$

The constraints **S0**, **S1**, **S2** and **S3** are as in the Base Problem. Constraint **S4** requires that the fractions of time spent receiving data from the parent and executing at the node be bounded by 1. Note that constraint **S2** for $k = 0$ is implied by **S4**, and so we can discard it from the list of constraints for a Send-in-Parallel problem.

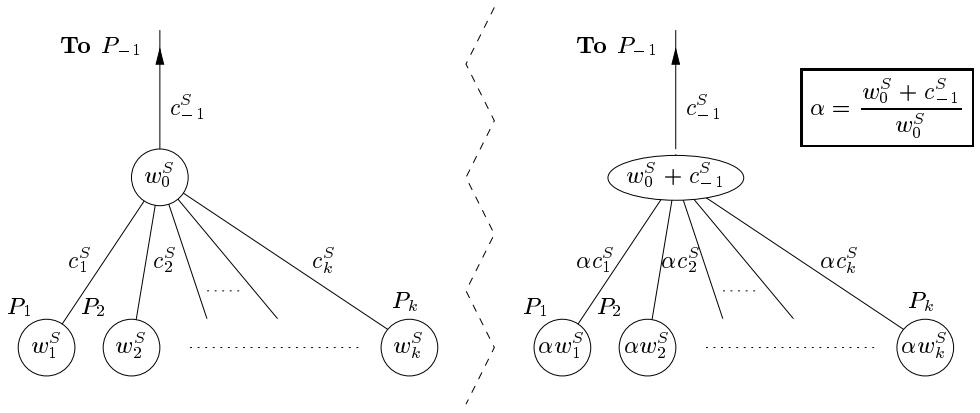

Figure 12: Reduction from Send-in-Parallel to Base.

We show that we can convert a Work-in-Parallel Problem to a Base Problem by adding the time spent receiving a task to the time needed to execute a task on the node, and multiplying each child's communication and work times by a carefully chosen constant.

Proposition 7 *Given a Send-in-Parallel Problem, there is an equivalent Base Problem whose solution is easily modified to give the solution to the Work-in-Parallel Problem.*

Proof Let $\alpha = \frac{w_0^S + c_{-1}^S}{w_0^S}$. We define $c_{-1}^B = c_{-1}^S$, $w_0^B = w_0^S + c_{-1}^S$, and for $1 \leq i \leq k$, $c_i^B = \alpha c_i^S$ and $w_i^B = \alpha w_i^S$ (see Figure 12).

We can convert between work allocations in the two different models using the following formulas: define $R_{-1}^B = R_{-1}^S$, $R_0^B = R_0^S + (1 - \frac{1}{\alpha}) \sum_{i=1}^k R_i^S$, and for $1 \leq i \leq k$, define $R_i^B = \frac{R_i^S}{\alpha}$. Note that $1 - \frac{1}{\alpha} = 1 - \frac{w_0^S}{w_0^S + c_{-1}^S} = \frac{c_{-1}^S}{w_0^S + c_{-1}^S}$, which we use later in the proof.

Rewriting constraint **S0** gives $R_{-1}^B = R_0^B - (1 - \frac{1}{\alpha}) \sum_{i=1}^k \alpha R_i^B + \sum_{i=1}^k \alpha R_i^B = R_0^B + \frac{1}{\alpha} \sum_{i=1}^k \alpha R_i^B = \sum_{i=0}^k R_i^B$, which is constraint **B0** for a base problem.

Rewriting constraint **S1**, we obtain $R_{-1}^B \leq \frac{1}{c_{-1}^B}$, which is constraint **B1**.

Similarly, rewriting constraint **S3** gives us $\sum_{i=1}^k \alpha R_i^B \frac{c_i^B}{\alpha} = \sum_{i=1}^k R_i^B c_i^B \leq 1$, which is **B3**.

Rewriting constraint **S2** for $1 \leq i \leq k$ produces $\alpha R_i^B \leq \frac{\alpha}{w_i^B}$, which, after dividing both sides by α, is **B2** for $1 \leq i \leq k$.

Finally, we will rewrite constraint **S4** to gives us the missing case of constraint **B2** for $k = 0$.

Rewriting the left hand side of **S4**:

$$\begin{aligned}
R_0^S w_0^S + R_{-1}^S c_{-1}^S &= R_0^S w_0^S + c_{-1}^S (R_0^S + \sum_{i=1}^k R_i^S) \\
&= R_0^S w_0^S + R_0^S c_{-1}^S + c_{-1}^S (\frac{w_0^S + c_{-1}^S}{w_0^S + c_{-1}^S}) \sum_{i=1}^k R_i^S \\
&= (w_0^S + c_{-1}^S) R_0^S + (w_0^S + c_{-1}^S) (\frac{c_{-1}^S}{w_0^S + c_{-1}^S}) \sum_{i=1}^k R_i^S \\
&= (w_0^S + c_{-1}^S)(R_0^S + (1 - \frac{1}{\alpha}) \sum_{i=1}^k R_i^S) \\
&= w_0^B R_0^B
\end{aligned}$$

Thus, **S4** is equivalent to $w_0^B R_0^B \leq 1$, i.e. $R_0^B \leq \frac{1}{w_0^B}$, which is **B2** for $k = 0$.

We have shown that the constraints of the original Send-in-Parallel Problem exactly correspond to those of the new Base Problem, so the solution of the Base Problem yields the solution to the Send-in-Parallel Problem.

There is one final detail to attend to: we must show that the solution $\{R_i^S\}$ computed from $\{R_i^B\}$ has $R_i^S \geq 0$ for $i = -1, 0, 1, ...k$. Unfortunately, this isn't

necessarily true; in fact, there may be many ways to maximize $\sum_{i=0}^{k} R_i^B$, and some of these ways will produce a negative value for R_0^S. However, we claim (a) that one can maximize $\sum_{i=0}^{k} R_i^B$ by keeping the node's processor (in the Base model) fully busy, i.e. that there is a maximum solution with $R_0^B = 1/w_0^B$, and (b) that this will provide a solution to the Send-in-Parallel Problem with $R_0^S \geq 0$.[6]

To establish claim (a), we note that since $w_0^B = w_0^S + c_{-1}^B \geq c_{-1}^B$, there is sufficient bandwidth from the node to its parent to keep the processor busy, i.e. constraint **B1** can be satisfied even if we set $R_0^B = 1/w_0^S$. Furthermore, we argue that in the Base model, it cannot hurt to keep the processor as busy as possible — either the link to the parent is the bottleneck (in which case it doesn't matter whether work is assigned to the processor or to a child), or the bottleneck is the amount of work that can be offloaded to the children (in which case, maximizing the work on the processor is necessary.)

To prove claim (b), note that once we set $R_0^B = 1/w_0^B$, we can rewrite:

$$R_0^S = R_0^B - (1 - \frac{1}{\alpha}) \sum_{i=1}^{k} R_i^S = \frac{1}{w_0^B} - (1 - \frac{1}{\alpha}) \sum_{i=1}^{k} \alpha R_i^B = \frac{1}{w_0^B} - (\alpha - 1) \sum_{i=1}^{k} R_i^B$$

Since $R_0^B = 1/w_0^B$, constraint **B1** tells us $\sum_{i=1}^{k} R_i^B \leq 1/c_{-1}^B - 1/w_0^B = (w_0^B - c_{-1}^B)/c_{-1}^B w_0^B$.

We can also rewrite $\alpha - 1 = c_{-1}^S / w_0^S = c_{-1}^B / (w_0^B - c_{-1}^B)$. Making these two substitutions in the above formula, we obtain:

$$R_0^S \geq \frac{1}{w_0^B} - \frac{c_{-1}^B}{(w_0^B - c_{-1}^B)} (\frac{w_0^B - c_{-1}^B}{c_{-1}^B w_0^B}) = \frac{1}{w_0^B} - \frac{1}{w_0^B} = 0.$$

showing that R_0^S is non-negative and completing the proof.

■

5 A more general communication cost model

So far, we have been considering a moderately simple model of communication costs. In particular, we have used a single parameter, $c_{i,j}$, to represent the cost of communication between nodes v_i and v_j. This single parameter serves three roles: first,

[6] It is obvious that the other R_i^S will be non-negative whenever the corresponding R_i^B are.

it constrains how much data can travel between the two nodes to be at most $1/c_{i,j}$ tasks per unit time. Second, it represents how much of a shared resource at node v_i is consumed by the communication. What this "shared resource" is depends on the model of the processor as described in Section 4; after the problem has been transformed to a Base Problem, it will be the limit given by the constraint **B3**. Finally, it represents the use of a shared resource on the other end of the commuication channel, node v_j.

There is no particular reason why, in a heterogeneous network, these limits will be the same. For instance node v_i may have a powerful communication subsystem that can handle communication tasks with little interference to other tasks, while node v_j may be less powerful.

The LogP model of communication [15] provides three communication parameters. The *latency* L is the elapsed time from when a message is put on one end of the communication channel to the time that it is received at the other. The *overhead* o is how much time the sending or receiving processor must spend dealing with the message. Finally, the *gap* g is how much time must pass between successive messages being sent on the channel. The gap actually represents a bandwidth limitation; the number of messages per unit time that can be sent is limited to $1/g$.

We will adapt this model for our purposes. Since we are only considering the "steady state" performance of the system, we do not need to concern ourselves with the latency L. However, to reflect the heterogeneous nature of the system, there will be two (possibly different) overhead parameters, representating the interference on the processors at either end of the channel.

With respect to a given node, we will use g_i to represent the gap of the channel from the node to its i-th child and g_{-1} for the gap of the channel to its parent. Similarly, we will use o_i and o_{-1} to represent the overhead of these communications on the node's processor.

We will show how the mechanisms developed earlier in this paper can handle this more general communication model. As in section 4, we let R_0 represent the number of tasks per unit time computed at the node iteslf, R_i be the rate of sending tasks to the i-th child, and $R_{-1} = \sum_0^k R_i$. However, we give these numbers a slightly different meaning in this new model: they will represent how much work the the subtree rooted at the node could do, assuming that there were infinite bandwidth from the node to its parent. In other words, we will not consider the constraint $R_{-1} \leq 1/g_{-1}$ when we

find a maximum solution for a subtree; instead, the constraint will be applied when dealing with the node's parent.[7]

At a given node, we have the following constraints:

Gap-Overhead Problem: Maximize $\sum_{i=0}^{k} R_i$, **subject to**

(G0) $R_{-1} = \sum_{i=0}^{k} R_i$

(G1) $R_i g_i^G \leq 1$ for $1 \leq i \leq k$

(G2) $R_i \leq \frac{1}{w_i^G}$ for $0 \leq i \leq k$

(G3) $R_0 w_0^G + \sum_{i=1}^{k} R_i o_i^G + R_{-1} o_{-1}^G \leq 1$

Constraint **G0** ensures, as always, that each task communicated to a node is either processed by the node or delegated to a child. **G1** ensures that the bandwidth limit to each child is respected. Note that by the earlier comment, we don't include the corresponding constrain for the channel to the parent. Constraint **G2** asserts that the processing speed limits are observed. Finally, **G3** limits the work that can be done on the node's processor, taking into account the overhead on the processor of each communication.

We now show that we can convert the Gap-Overhead Problem to the fully sequential model $\mathcal{M}(r, s, w)$.

Proposition 8 *Given an instance of the Gap-Overhead Problem, there is an equivalent No-Overlap Problem instance with* $\delta = 1$ *(i.e., an instance in the fully sequential model).*

Proof Given a Gap-Overhead instance, we let $c_{-1}^N = o_{-1}^G$, $w_0^N = w_0^G$, and for $1 \leq i \leq k$, let $c_i^N = o_i^G$ and $w_i^N = \max(g_i^G, w_i^G)$.

Constraint **G0** is the same as **N0** of the No-Overlap Problem.

The $i = 0$, cases of constraint **G2** and **N2** are the same since $w_0^G = w_0^N$. For $1 \leq i \leq k$, constraints **G1** and **G2** can be combined as $R \leq 1/max(g_i^G, w_i^G) = 1/w_i^N$ for $1 \leq i \leq k$. This is the corresponding constraint **N2**.

After the variable substitution, constraint **G3** becomes constraint **N3** when $\delta = 1$.

Finally, notice that constraint **N1** (that $R_{-1} \leq 1/c_{-1}^N$) is implied by constraint **N3** (when $\delta = 1$).

[7] It would have been less natural to use this interpretation in Section 4, where the single commuication parameter could intefere in both the parent's and the child's processing. Now that communication costs are broken into components, it is simpler to consider the bandwidth limitation on a channel only when processing the endpoint closer to the tree's root.

Thus, the constraints of the Gap-Overhead Problem exactly correspond to the constraints of the fully sequential problem, so the Gap-Overhead Problem can be solved by the techniques of section 4.3. ■

Intuitively, this result can be interpreted as follows: In order to solve a Gap-Overhead problem, we first replace the compute time of each child by the max of its compute time (w_i) and the communication time (g_i). Clearly, we can't get a child to work faster than that limit. Next we treat it as a fully sequential problem, where the overheads (both o_{-1} and the o_i's) play the role of the communication times. This is natural, since in the fully sequential problem, the node's processor is interrupted by the overhead of communicating either to its parent or to any child. The solution to the fully sequential problem gives the maximum work the node can perform, ignoring the constraint $R_{-1} \leq 1/g_{-1}$, which will be handled when this node is considered as a child.

6 Simulations

6.1 Demand Driven Heuristics

In order to experimentally demonstrate the effectiveness of our approach, we tested three different heuristics. The three heuristics use a similar, demand-driven approach; the main difference between them is the amount of information provided by Proposition 1 that they use. All three heuristics use the following outline: each node is initally given one task, except for the root which has 1000 tasks. Then, for each time step, for each node,

- If the node's processor is free and there is an unexecuted a task, then start executing the task and request another task from the parent.
- For each child, if there is a new request from the child, put the request into a queue, and request another task from the parent.
- If communication with a child is possible (meaning there is an outstanding request, there is a task available, and the communication channel is free), then start sending the task to the child.

We process the nodes from the leaves up to the root of the tree; thus in our simulations, all requests are propagated up to the root in the same cycle that they are generated.

The three heuristics are described below:

- Heuristic 1: This heuristic uses all the processors. Requests for tasks are satisfied in first-come, first-serve order. This heuristic uses no information from Proposition 1.

- Heuristic 2: This heuristic only uses the processors which are assigned tasks in the optimal solution (Proposition 1). The requests are satisfied in first-come, first-serve order. Note that this heuristic may give more tasks to the $(p+1)$-th child than would be executed under the optimal schedule. (Recall that in the optimal schedule, this child should be idle part of the time to ensure that all children with faster communication times are kept fully busy.)

- Heuristic 3: This heuristic uses the same processors as the second heuristic, but it satisfies requests from the $(p+1)$-th child only if there are no other outstanding requests. Note that this child still could receive more tasks than in the optimal solution, but it is less likely than with the second heuristic.

All three heuristics are demand-driven and fully dynamic. Indeed, the only information that we use from Proposition 1 are choosing which processors will be fully active, partly idle, and fully idle. These sets will not be (in general) affected by small changes in machine parameters. Therefore, the heuristics should not be affected by small dynamic changes in processor speeds and/or link bandwidths.

6.2 Simulation Results

We present the results of the heuristics for two different sets of tree graphs. All the simulations have been performed with 100 randomly generated trees. The first set of simulations corresponds to a simple fork graph — a root node with between 2 and 6 children. The results are displayed in Figure 13. The second set of simulations corresponds to more general trees. For these simulations, each node has at most 5 children and at most 10 nodes have children. The results are displayed in Figure 14. For all these simulations, all the nodes (except the root of the tree) own exactly one task at the beginning of the execution. The root of the tree owns 1000 tasks at the beginning. Each figure displays the ratio between the time necessary to process 1000 tasks with a given heuristic and the optimal steady state time given by Proposition 1. The dashed line represents the ratios obtained with the first heuristic, the dotted line represents the ratios obtained with the second heuristic and the solid line represents the ratios obtained with the third heuristic. Each point on the x-axis represents a different randomly-generated tree. They have been sorted according to the ratio obtained with the first heuristic.

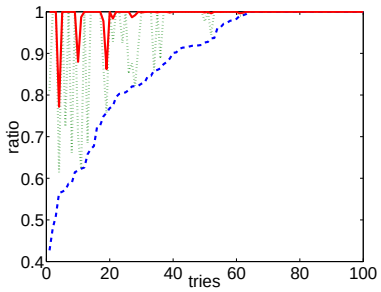

Figure 13: Simulations with a simple fork graph

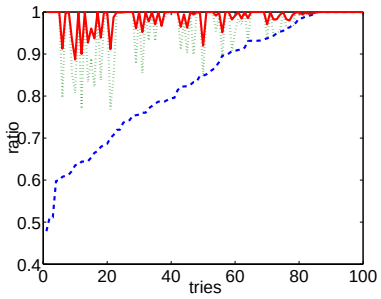

Figure 14: Simulations with a more general tree

	mean value Fork Graph	minimal value Fork Graph	mean value General Tree	minimal value General Tree
Heuristic 1	0.83	0.48	0.86	0.43
Heuristic 2	0.96	0.77	0.96	0.61
Heuristic 3	0.98	0.88	0.99	0.77

Figure 15: Minimal and mean values of the ratio for the different heuristics

Both Figures 13 and 14 illustrate the value of taking into account the static information given by Proposition 1, even in the design of fully dynamic algorithms. The mean and minimal values of the ratios for the different heuristics are displayed in Table 15.

6.3 Analysis of a compute-limited case

Let us consider the fork graph depicted in Figure 16, with a root P_0 and three children P_1, P_2 and P_3. With the notations of Proposition 1,

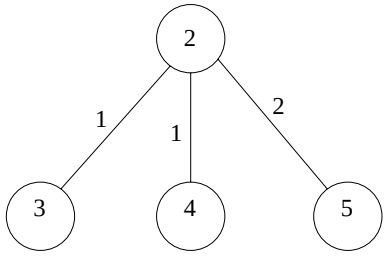

Figure 16: Fork graph with three children

$$\sum_{i=1}^{3} \frac{c_i}{w_i} = \frac{1}{3} + \frac{1}{4} + \frac{2}{5} = \frac{59}{60} < 1.$$

Thus, all the processors are kept fully active in the optimal solution, and all three heuristics give the same results. The ratio is in this case equal to 0.997. The small difference between 0.997 and 1 is related to the fact that it takes a few steps for the demand-driven algorithm to reach the steady state.

6.4 Analysis of a bandwidth-limited case

Let us consider the fork graph depicted in Figure 17, with a root P_0 and 3 children P_1, P_2, P_3, P_4 and P_5. Two children with slow links have been added. In this case,

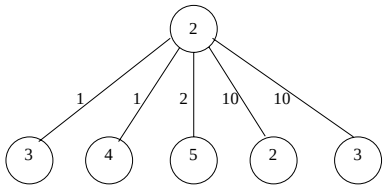

Figure 17: Fork graph with five children

$$\sum_{i=1}^{4} \frac{c_i}{w_i} = \frac{59}{60} + \frac{2}{10} = \frac{71}{60} > 1,$$

so that in the optimal solution given by Proposition 1, P_4 is partly idle and P_5 is kept fully idle. The ratios obtained by heuristics 1, 2 and 3 are respectively 0.55, 0.61 and 0.88. With the first heuristic, processors P_0, P_1, P_2, P_3, P_4 and P_5 respectively process $70\%, 6\%, 6\%, 6\%, 6\%$ and 6% of the tasks. The last two children process just as many tasks as the other children. In fact, in this case, the root of the tree is sending tasks to its children all the time and, since the last two children have small processing times, they request tasks just as often as the other children. With the second heuristic, processors P_0, P_1, P_2, P_3 and P_4 (P_5 is not used any more) respectively process $64\%, 9\%, 9\%, 9\%$ and 9% of tasks. The same phenomenon occurs, but in this case, since P_5 is not used, the results are slightly better. With the third heuristic, processors P_0, P_1, P_2, P_3 and P_4 (P_4 is given low priority and P_5 is not used at all) respectively process $44\%, 22\%, 18\%, 14\%$ and 2% of tasks. This solution is much more balanced and is closer to the optimal solution given by Proposition 1, where the nodes respectively process 39%, 26%, 20%, 16% and 0.1% of tasks. Nevertheless, we are far from the optimal solution (the ratio is 0.88) because the last child still processes too many tasks and has a very slow link.

One way to overcome this problem is to give more tasks to each processor at the beginning of the execution. The values of the ratio become 0.57, 0.62, 1.001 if the children own 4 tasks at the beginning of the execution (instead of 1). The fact that the time necessary to process 1000 tasks with the heuristics may be smaller than in the optimal solution is not surprising since all the children are given 4 tasks "for free" (without communication) at the beginning of the execution. In this case, with

the third heuristic, the different nodes respectively process 39%, 26%, 19%, 16% and 0.6% of tasks, which is very close to the optimal solution.

6.5 Simulations with larger initial distributions

As we just saw, giving more tasks at the beginning of the execution is effective in making a simulation of 1000 tasks using the third heuristic match the theoretical steady-state performance. An alternate method would be to simulate far more than 1000 tasks. Unfortunately, we do not yet know an efficient way to initialize the simulations to enter steady-state immediately.

Figure 18 and 19 (corresponding respectively to Figures 13 and 14) display the results when 4 tasks are given to each processor (instead of 1), and Table 20 summarizes the mean and minimal values of the ratio for the different heuristics. In this case, we note that the third heuristic is almost optimal. Also notice that in Figure 18, the simulations of the first heuristic are sometimes significantly better than the optimum. This artifact is a result of our giving four "free" tasks to each of several processors that aren't even used in the optimum solution.

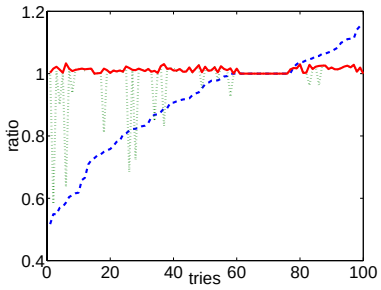

Figure 18: Simulations with a simple fork graph

7 Related work

Mapping and scheduling tasks onto heterogeneous platforms has received considerable attention in recent years (see the survey papers of Berman [7] and Feitelson [17]).

In contrast to our work, many related works use the execution time of the application to guide its task allocation. A well-known example of a high-level scheduling and load-balancing tool is AppLeS, which provides application-level scheduling agents

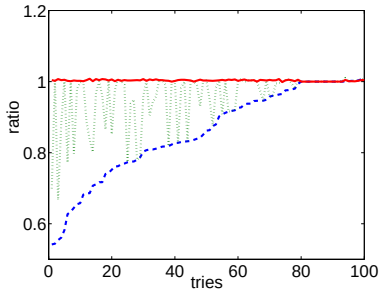

Figure 19: Simulations with a more general tree

	mean value Fork Graph	minimal value Fork Graph	mean value General Tree	minimal value General Tree
Heuristic 1	0.86	0.54	0.91	0.51
Heuristic 2	0.96	0.67	0.98	0.58
Heuristic 3	1.00	1.00	1.01	1.00

Figure 20: Minimal and mean values of the ratio for the different heuristics

whose goal is to minimize the application's execution time on dynamically changing, heterogeneous systems [26, 8]. AppLeS agents utilize the Network Weather Service [35] to monitor the varying performance of resources potentially available to their applications, and use this information to predict the resources available at the time the application will be scheduled. Another closely related problem is to determine the best allocation of tasks to workers (using a Master-Worker paradigm), given a fixed time bound. This is considered, for instance, by Beaumont et al [6]. In our terms, their model corresponds to scheduling a fork-join node in a No-Overlap model $\mathcal{M}(r, s, w)$where the parent (the Master) cannot perform any computation.

Most related work only considers a single level of heterogeneity. The work of [12] uses a combined compile-time/runtime approach to create customized load balancing strategies for an application to minimize its execution time on a single cluster of workstations. In [11], a purely static, architecture-conscious approach is considered for load-balancing parallel loops which may have different amount of work in each iteration, on a (possibly) heterogeneous network of workstations. This work also considers the effects of limited memory size. In Andonie et al. [2], a dynamic programming approach is used to minimize the execution time of independent tasks (for

training distributed backpropagation neural networks) on a heterogeneous network of workstations.

There are some works that (like ours) consider a multiple-leveled network; for instance [21] is an empirical evaluation of several scheduling strategies (round robin, first come first serve, etc) when applied hierarchically for clusters of subclusters.

Some approaches to minimizing execution time are purely static. Algorithms for statically partitioning a large rectangular iteration space so as to load-balance the work of different-speed processors are given in Crandall and Quinn [14] and Kaddoura, Ranka and Wang [22]. Proposals for the extension of the ScaLAPACK library to heterogeneous processors are given by Barbosa, Tavares and Padilha [3], Kalinov and Lastovetky [23], and Beaumont et al [4, 5]. All these papers target a specific class of algorithms where processors are assigned rectangular tiles of a large iteration space; for each processor, computation costs are proportional to its tile surface while communication costs are proportional to its tile perimeter.

Other approaches — notably the various self-scheduling techniques [25, 20, 19, 30] — use dynamic heuristics. All but [19] were developed for homogeneous systems. Because our approach does not give equal priority to all requests, we can do better in steady-state than self-scheduling, as shown in our experimental results.

Several theoretical papers deal with complexity results for parallel machine problems with a server. In this model, heterogenous tasks are allocated by the server one at a time on the homogeneous parallel machine. NP-completeness results are given in [18, 24, 9] and guaranteed approximations in [27]. In constrast, our work consider the allocation of homogenous tasks on a heterogeneous system.

The work most closely related to ours is Shao's thesis [33]. He suggests using the Maximum-Flow algorithm on the graph (not necessarily a tree) representing heterogeneous resources to maximize steady-state throughput of Master-Worker applications. In contrast, our work obtains a simple, closed-form formula for maximizing steady-state, which in turn gives insight into the construction of a dynamic algorithm.

8 Conclusion and discussion

In this paper, we considered the problem of allocating a large number of independent, equal-sized tasks to a heterogeneous collection of computing resources. We assume the data for the tasks is initially located on a single resource. Such problems arise in collaborative computing efforts.

This paper makes the following contributions to this problem:

- We suggest that it is natural to model the computing resources as a tree, whose nodes can have different speeds of computation and communication. Our model also allows nodes to have differing capabilities in terms of allowing overlap of computation and communication.
- For one particular model of overlap capabilities — the base model which allows computation and communication in two directions to overlap — we give an explicit solution to the problem of determing the allocation of tasks to nodes in the tree that maximizes the number of tasks executed per unit time in steady-state. This optimal solution is *bandwidth-centric*: tasks should be allocated to nodes in order of fastest communication time. This result may be counter-intuitive, it says the speed of the processors is irrelevant to choosing which processors to send tasks to.
- We show how other models of overlap capability can be reduced to the base model. The reductions work on a node-by-node basis, so it is simple to handle a heterogeneous system with nodes having not only differing speeds but also differing capabilities.
- We define a more general communication model which includes the overheads and gap of the LogP model, and showed how to obtain the best steady-state task allocation for this model as well.
- We present simulation results of demand-driven task allocation heuristics. These results show that our bandwidth-centric method, where tasks are allocated only to nodes with sufficiently fast communication times, obtains better results than allocating tasks to all processors on a first-come, first serve basis. Even better results are given by giving a low priority to the processors that, in the optimal solution, are idle part of the time.

Our model does not directly address the dynamically changing nature of the grid, nor does it consider the total execution time from sending out the first task to receiving the result from the very last. Although both of these are important considerations, this paper nonetheless suggests a simple and powerful heuristic: allocation of tasks at each node should be bandwidth-centric. In other words, given limited bandwidth, tasks should be allocated locally at each node with priority given to *how fast they can be communicated* from the node to a child, and not based on the computational speed of the child.

This heuristic can be incorporated into an autonomous and dynamically self-adjusting scheduling policy. Each node should have a finite buffer for unprocessed

tasks. The buffer should be large enough to smooth over "bursty" behavior, but if it too large, there may be longer start-up and finish-up times. When the number of tasks in the node's buffer falls below a critical level,[8] the node should request additional tasks from its parent node. Meanwhile, a node should prioritize the requests it receives from its children according to the time it takes to communicate with the child. The times used for prioritization are simply the times that the node measures for its side of the communication.

In a human example, suppose a manager can delegated tasks to Bob via US mail (which only requires 30 seconds to address an envelope), but delegating to Eve requires 4 minutes to operate a fax machine. Then the manager should always answer Bob's requests for more work before responding to Eve's, even though the mail to Bob may be slow, and regardless of whether Bob or Eve is the faster worker.[9]

Note that the decision about how many tasks to process in the node itself depends on the overlap model. In the Base Model, where computation does not intefere with communication, the node should assigns itself work with highest priority. When computation slows commuication down, then the transformations of section 4 describe how to assign a priority to local processing.

The strategy described above can adapt to dynamically changing network conditions. Each node periodically measures its bandwidth to its parent and children, and changes priorities accordingly. There are practical issues involved in estimating current communication costs from historical data; these issues have been investigated by the Network Weather Service [35]. A nice feature of our approach is that each node makes its decisions autonomously. The explicit solution we derived, which maximizes the throughput of the entire grid, does not require solving a global max flow problem (as suggested, for instance, by Shao [33].)

By ignoring startup costs and focusing on steady-state throughput, we were able to obtain a simple, explicit solutions to our optimization problem. However, it would be interesting to invesitgate the cost of startup, through theory, simulations, and actual experiments.

That said, we note that the question of whether it's better to maximize steady-state performance or to minimizing application execution time is a complex one. A system administrator may want to optimize steady-state performance, noting that (assuming the applications can adapt to dynamically changing conditions) while one

[8] The optimal value for this level should be the subject of more research. It will most likely depend, among other things, on the *latency* of each network connection, and thus cannot be answered using only the parameters of our model.

[9] Of course, if Bob is slower, there will be fewer requests from him. And it is important that Bob maintain a large enough backlog of work so that he is not idle while waiting for the mail delivery.

application is starting up or finishing, other applications are able to make use of these idle resources. What *does* matter to the administrator is that if the network is the bottleneck, then faster network resources should be used in preference to slower ones. Our solution has this feature. On the other hand, a user with only one application to run may only care about the turnaround time for that particular job. However, the turnaround issue is complex. If new jobs are created faster than the system can process them, then the system will get further and further behind in its work; asymptotically, the average turnaround time will be infinite. The best way to prevent this situation, or to make the backlog grow as slowly as possible, is to have the highest possible throughput. Thus in some situations, it can be argued that steady-state performance is all that matters.

There are many other ways our work could be extended. Future work should consider the addition of data dependences between tasks. It should also consider, instead of having a constant per-task communication cost, that sending a larger amount of work involves some savings in per-task communication costs. This would reflect applications (such as matrix multiplication) that have a favorable communication-to-computation ratio; however, it would require adding an additional parameter, memory capacity, to model each node. Another generalization would allow initial data to reside on several nodes, instead of a single node.

It is likely that many of these harder problems, particularly those that address the total application execution time, will be NP-complete. The bandwidth-centric approach of this paper may suggest good heuristics in these cases.

References

[1] B. Alpern, L. Carter, and J. Ferrante. Modeling parallel computers as memory hierarchies. In *Proceedings of the Working Conference on Massively Parallel Programming Models*. IEEE Computer Society Press, 1993.

[2] R. Andonie, A.T. Chronopoulos, D. Grosu, and H. Galmeanu. Distributed backpropagation neural networks on a PVM heterogeneous system. In *Parallel and Distributed Computing and Systems Conference (PDCS'98)*, pages 555–560. IASTED Press, 1998.

[3] J. Barbosa, J. Tavares, and A.J. Padilha. Linear algebra algorithms in a heterogeneous cluster of personal computers. In *9th Heterogeneous Computing Workshop (HCW'2000)*, pages 147–159. IEEE Computer Society Press, 2000.

[4] Olivier Beaumont, Vincent Boudet, Fabrice Rastello, and Yves Robert. Load balancing strategies for dense linear algebra kernels on heterogeneous two-dimensional grids. In *14th International Parallel and Distributed Processing Symposium (IPDPS'2000)*, pages 783–792. IEEE Computer Society Press, 2000.

[5] Olivier Beaumont, Vincent Boudet, Fabrice Rastello, and Yves Robert. Matrix-matrix multiplication on heterogeneous platforms. In *2000 International Conference on Parallel Processing (ICPP'2000)*. IEEE Computer Society Press, 2000.

[6] Olivier Beaumont, Arnaud Legrand, and Yves Robert. The master-slave paradigm with heterogeneous processors. Technical Report RR-2001-13, LIP, ENS Lyon, March 2001. Available at `www.ens-lyon.fr/LIP/`.

[7] F. Berman. High-performance schedulers. In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, pages 279–309. Morgan-Kaufmann, 1999.

[8] F. Berman and R. Wolski. TheAppLeS project: A status report. In *Proceedings of the 8th NEC Research Symposium*, 1997. Available at `http://www.gcl.ucsd.edu//hetpubs.html#AppLeS`.

[9] P. Brucker, C. Dhaenens-Flipo, S. Knust, S.A. Kravchenko, and F. Werner. Complexity results for parallel machine problems with a single server. Technical Report Reihe P, No. 219, Fachbereich Mathematik Informatik, Universität Osnabrück, 2000.

[10] P. Chrétienne, E.G. Coffman Jr., J.K. Lenstra, and Z. Liu, editors. *Scheduling Theory and its Applications*. John Wiley and Sons, 1995.

[11] Michal Cierniak, Mohammed J. Zaki, and Wei Li. Compile-time scheduling algorithms for heterogeneous network of workstations. *The Computer Journal*, 40(6):356–372, 1997.

[12] Michal Cierniak, Mohammed J. Zaki, and Wei Li. Customized dynamic load balancing for a network of workstations. *Journal of Parallel and Distributed Computing*, 43:156–162, 1997.

[13] James Cowie, Bruce Dodson, R.-Marije Elkenbracht-Huizing, Arjen K. Lenstra, Peter L. Montgomery, and Joerg Zayer. A world wide number field sieve factoring record: on to 512 bits. In Kwangjo Kim and Tsutomu Matsumoto, editors,

Advances in Cryptology - Asiacrypt '96, volume 1163 of *LNCS*, pages 382–394. Springer Verlag, 1996.

[14] P.E. Crandall and M.J. Quinn. Block data decomposition for data-parallel programming on a heterogeneous workstation network. In *2nd International Symposium on High Performance Distributed Computing*, pages 42–49. IEEE Computer Society Press, 1993.

[15] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. v. Eicken. Logp: Towards a realistic model of parallel computation. In *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM Press, 1993.

[16] Entropia. URL: `http://www.entropia.com`.

[17] D.G. Feitelson. High performance cluster computing. volume 1: Architecture and systems. In R. Buyya, editor, *The Grid: Blueprint for a New Computing Infrastructure*, pages 519–533. Prentice Hall PTR, 1999.

[18] N. Hall, C.N. Potts, and C. Sriskandarajah. Parallel machine scheduling with a common server. *Discrete Applied Mathematics*, 102:223–243, 2000.

[19] Susan Flynn Hummel, Jeanette Schmidt, R. N. Uma, and Joel Wein. Load-sharing in heterogeneous systems via weighted factoring. In *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 318–328, June 1996.

[20] Susan Flynn Hummel, Edith Schonberg, and Lawrence E. Flynn. Factoring: A method for scheduling parallel loops. *Communications of the ACM*, 35(8):90–101, 1992.

[21] H. A. James, K. A. Hawick, and P. D. Coddington. Scheduling independent tasks on metacomputing systems. Technical Report DHPC-066, University of Adelaide, Australia, 1999.

[22] M. Kaddoura, S. Ranka, and A. Wang. Array decomposition for nonuniform computational environments. *Journal of Parallel and Distributed Computing*, 36:91–105, 1996.

[23] A. Kalinov and A. Lastovetsky. Heterogeneous distribution of computations while solving linear algebra problems on networks of heterogeneous computers.

In P. Sloot, M. Bubak, A. Hoekstra, and B. Hertzberger, editors, *HPCN Europe 1999*, LNCS 1593, pages 191–200. Springer Verlag, 1999.

[24] S.A. Kravchenko and F. Werner. Parallel machine scheduling problems with a single server. *Mathematical Computational Modelling*, 26:1–11, 1997.

[25] Clyde P. Kruskal and Alan Weiss. Allocating independent subtasks on parallel processors. *IEEE Transactions on Software Engineering*, 11(10):1001–1016, October 1985.

[26] Grid Computing Laboratory. Apples. URL: `http://apples.ucsd.edu`.

[27] H. Lee and M. Guignard. A hybrid bounding procedure for the workload allocation problem on parallel unrelated machines with setups. *Journal of the Operational Research Society*, 47:1247–1261, 1996.

[28] C.E. Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, 34(10):892–901, 1985.

[29] B. Lowenkamp, N. Miller, D. Sutherland, T. Gross, P. Steenkiste, and J. Subhlok. A resource query interface for network-aware applications. In *Proceedings of the Seventh International Symposium on High Performance Distributed Computing*, 1998.

[30] Constantine D. Polychronopoulos and David J. Kuck. Guided self-scheduling: A practical scheduling scheme for parallel supercomputers. *IEEE Transactions on Computers*, C-36(12):1485–1495, 1987.

[31] Prime. URL: `http://www.mersenne.org`.

[32] SETI. URL: `http://setiathome.ssl.berkeley.edu`.

[33] Gary Shao. *Adaptive scheduling of master/worker applications on distributed computational resources.* PhD thesis, University of California at San Diego, May 2001.

[34] Gary Shao, Fran Berman, and Rich Wolski. Using effective network views to promote distributed application performance. In Hamid R. Arabnia, editor, *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*. CSREA Press, 1999.

[35] R. Wolski, N. T. Spring, and J. Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(10):757–768, 1999.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399